Institutionen för datavetenskap
Department of Computer and Information Science

Final thesis

Extending CompactRIO connectivity through Anybus CompactCom

by

Emil Martinsson

LIU-IDA/LITH-EX-G--14/005--SE

2014-03-19

Linköpings universitet

Linköpings universitet
SE-581 83 Linköping, Sweden

Linköpings universitet
581 83 Linköping

Linköpings universitet
Institutionen för datavetenskap

Examensarbete

Anybus CompactCom modul för CompactRIO System

av

Emil Martinsson

LIU-IDA/LITH-EX-G--14/005--SE

2014-03-19

Handledare: Bogdan Tanasa
Examinator: Unmesh D. Bordoloi

Glossary

Term	Description
ABCC	Anybus CompactCom, a product concept developed by HMS which are communication solutions for fieldbus and industrial Ethernet [13].
Bluetooth	A wireless communication standard used to exchange data over short distances [1].
CAN	The serial bus CAN, Controller Area Network, developed for automotive applications allow nodes to send messages fast and secure [3].
CANopen	Communication protocol used in embedded control system [2].
CC-Link	Open network standard mainly used in industrial networks [4].
CIP	Common Industrial Protocol, CIP, is an upper layer used in e.g. DeviceNet network protocol. It standardizes the objects that can be specified inside the network so that the user knows what to access where [26].
ControlNet	Fieldbus used in industries where transmission of time-critical I/O data and messaging data are transmitted at high-speed [24].
CompactRIO	CompactRIO or cRIO is a system developed by National Instrument for controlling or observing industries applications.
DeviceNet	A network designed for industrial automation [25].
EtherCAT	Ethernet for Control Automation Technology extends the Ethernet technology by reducing the processing time of the Ethernet frames making it ideal for automation applications [5].
NI	National Instruments, company that develops cRIO systems and LabVIEW.
NI LabVIEW	LabVIEW is a graphical programming platform for engineers. It is used for measurements and control system.
PCB	Printed Circuit Board.
PDO	Process Data Object is used for delivering real time data between nodes in the CANopen protocol [9].
PROFIBUS	An application-independent communication protocol that is used in both process automation as motion control and safety related tasks [27].
RS232	Serial interface.
SDO	Service Data Object is used in CANopen as a direct access between two nodes to read or write to a certain object within the requested device [8].
USB	Universal Serial Bus, an interface mainly used to connect peripherals to computers as well as connecting devices together [29].
VI	Virtual Instruments, a program or subroutine in a LabVIEW environment.
XXh	Hexadecimal number, where XX refers to the specific number.

Table of Contents

Glossary 1
Table of Contents 3
Abstract 5
Preface 6

Chapter 1 Introduction 7

1.1 Background 7
1.2 Problem description 7
1.3 Goal 7
1.4 Approach 7
1.5 Thesis limitations 7
1.6 Thesis structure 8

Chapter 2 WireFlow, cRIO systems and ABCC 9

2.1 WireFlow AB 9
2.2 CompactRIO systems 9
2.2.1 Background 9
2.2.2 Example applications 9
2.2.3 cRIO modules 10
2.2.4 NI LabVIEW 10
2.3 Anybus CompactCom 11
2.3.1 Fieldbuses and Ethernets 11
2.3.2 Active and passive modules 11
2.3.3 Hardware design 11
2.3.4 Communication 12
2.3.5 The Anybus state machine 16
2.3.6 The object model 17
2.3.7 Application Data Instances 18

Chapter 3 Approach 21

3.1 Starter kit 21
3.2 LabVIEW communication 21
3.3 Modules 22
3.4 CANopen 22
3.5 Full testing of Anybus SDK 23
3.5.1 Anybus SDK 23
3.5.2 CANopen communication to Anybus SDK 23
3.5.3 Conclusion of tests towards Anybus SDK 24
3.6 LabVIEW SDK 24
3.6.1 Conclusion of the LabVIEW SDK 26
3.7 LabVIEW SDK and ABCC module in cRIO chassis 26
3.7.1 PCB design notes 27
3.7.2 Fully functional prototype 28

Chapter 4 Evaluation 30

4.1 Requirements 30
4.1.1 cRIO modules 30
4.1.2 ABCC modules 31
4.1.3 Conclusion of requirements 33
4.2 How to fit a ABCC module inside a cRIO module 34
4.2.1 cRIO module components 34

4.2.2 ABCC module components 34
4.2.3 CompactFlash connector 35
4.2.4 Front panel of cRIO module 35
4.2.5 Alignment of ABCC module on cRIO module PCB 35
4.2.6 Creating a prototype PCB 36
4.2.7 Conclusion 36
4.3 cRIO interface together with the ABCC module 36
4.3.1 Object hierarchy 37
4.3.2 Conclusion 37
4.4 Market research 37
4.4.1 Conclusion 38
4.5 Expanding the LabVIEW SDK 38
4.5.1 Object support 38
4.5.2 Still to be implemented: 39
4.5.3 Real-life scenario 39
References 41

Abstract

This thesis report describes an exploratory work for a network communication modules to be used together with the CompactRIO system, a system used to control or monitor industrial machinery. This thesis was carried out by integrating the existing ABCC module series, which is a communication solution that can be used to control or observe industrial machinery, into a CompactRIO module. The report describes how the implementation has been carried out and describes the parts that were implemented and give the company, WireFlow AB, a good basis for future development.
The result shows that it is possible to integrate the ABCC modules in cRIO both in terms of hardware and software integrating. The thesis delivered a working cRIO prototype to the company together with a driver developed in LabVIEW.

Gothenburg, Sweden, March 2014

Emil Martinsson

Preface

This thesis report is an exploratory work for an IT company to be used as a basis for a new product development. I would like to thank all the personnel at WireFlow AB for making this thesis possible and I would also like to thank my supervisor Bogdan Tanasa and examiner Unmesh Bordoloi at Linköping University.

Gothenburg, Sweden, March 2014

Emil Martinsson

Chapter 1
Introduction

This thesis was carried out as a final project at candidate level for a three year degree in Computer science at Linköping University.

1.1 Background

As technology moves forward and changes from day to day, the need for adoption in industry networks is essential. Being able to adapt or expand systems is vital for companies to keep their system up to date. In today's market there are a lot of different types of fieldbuses, e.g. CC-Link, and Ethernet standards, e.g. EtherCAT, available for companies to use when expanding or renewing their current system layout. Making industrial control- and monitoring systems such as CompactRIO, or cRIO, able to communicate with many different networking standards is of high importance for the cRIO users.

1.2 Problem description

For a company that wants to expand their current product portfolio; is it possible to integrate an already made communications solutions in a cRIO module, taking full use of the ready-made product? The problem concerns, both the physical and software layout of this potential new product.

1.3 Goal

The goal of the thesis is to deliver a thorough pre-study to WireFlow in order to help them evaluate the potential of their new product. We have divided the thesis goal into four main objects:

- Will there physically be enough space in the cRIO module for the ABCC module, including extra needed electronics?
- Will the ABCC module meet the power requirements for cRIO modules?
- Are there any similar products available in the market today?
- How will the software interfaces between ABCC and cRIO work together?
    - In order to achieve this, the driver running the Anybus CompactCom must be translated into the programming language that the CompactRIO uses, LabVIEW.

1.4 Approach

This thesis has been carried out at WireFlow AB, located in Gothenburg. WireFlow is an expanding engineering company that develops both software and hardware for National Instruments CompactRIO systems. During this thesis the focus has been on both the software and hardware development of a new module for the CompactRIO system, making it available to communicate with almost any type of fieldbus on today's market, which was done by integrating the existing ABCC module in the CompactRIO.

1.5 Thesis limitations

This thesis has been limited in certain aspects since it was clear that covering every potential angle would be far too great for the duration of this thesis. Therefor the thesis has been focus on a specific

communication solution, CANopen, with mostly the software part of the solution although some hardware solutions are covered as well.

1.6 Thesis structure

The report is structured as:

- Chapter two gives the reader a background into WireFlow, cRIO systems and the structure of the Anybus CompactCom
- Chapter three explains the approach of the thesis
- Chapter four evaluates the result of the thesis

Chapter 2
WireFlow, cRIO systems and ABCC

This chapter will give a background to the company and to the two systems that is being used in this thesis.

2.1 WireFlow AB

WireFlow is an engineering company that works closely with National Instrument (NI). WireFlow develops both hardware and software products for NI's systems, such as CompactRIO. The company was founded three years ago and has today four employees located in Gothenburg. Since the start they have developed three modules for the CompactRIO series and also USB dongles for securing LabVIEW code [30].

2.2 CompactRIO systems

This part explains the main concept behind the CompactRIO systems developed by National Instruments (NI) and also gives a brief background to the programming language LabVIEW.

2.2.1 Background

CompactRIO, or cRIO, is a control system developed by National Instruments. The system consists of three main parts; a reconfigurable FPGA chassis, I/O modules and an embedded controller. The language for programming cRIO is National Instruments own LabVIEW [23]. cRIO systems are used to control or observe industrial applications, all from machinery to fire-alarms. They can also be used to simulate systems when testing products.

2.2.2 Example applications

The CompactRIO systems are used widely around the world e.g. mounted on an oil well pump to monitoring the system under harsh environments or to control hydraulic valves with great accuracy while lifting heavy concrete trays.

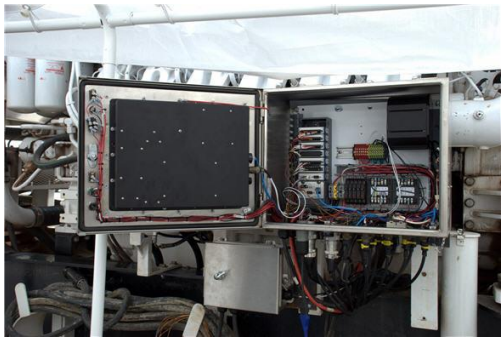


Figure 1 - CompactRIO system mounted on an oil well pump

Figure 2 - CompactRIO system used to lift and control a concrete tray with high accuracy

2.2.3 cRIO modules

cRIO modules are used together with cRIO chassis in industries for controlling or observing real-time or simulated environments.

Figure 3 - cRIO module [19]

Figure 4 - 4 slot cRIO chassis NI-9075 [17]

NI, who developed the cRIO system, have developed some modules for their cRIO chassis available today however they allow third-party developers, such as WireFlow, to develop modules for their cRIO chassis. The third-party developer can then send the design to NI for inspection and feedback.

2.2.4 NI LabVIEW

LabVIEW is a graphical orientated programming language that is used for developing NI related products [22]. LabVIEW is developed by National Instrument and is used in businesses around the world. In LabVIEW the user creates functions in VI's. A VI, or Virtual Instrument, is a part of a code that the user creates. Since LabVIEW is a graphical programming language no rows of code are written, instead code is created by placing functions inside of a VI. Each VI can be connected to other VI's making them sub functions to a bigger system as explained in the picture below. Each VI holds both a block diagram and a front panel. During runtime the user can interact with controllers on the front panel whereas the block diagram holds the code.

Figure 5 – Simple block diagram, a VI, holding three subVI's

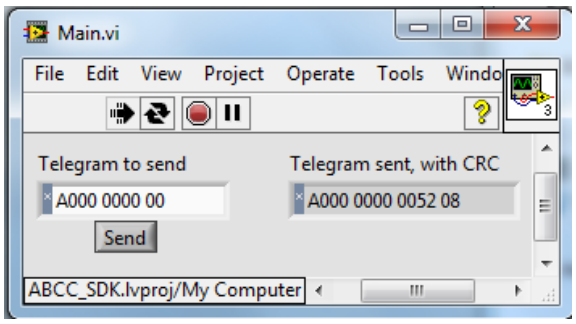


Figure 6 - Simple LabVIEW front panel

In LabVIEW the user make use of the sequence flow structure of the program meaning that each function is sequentially run from left to right. This makes it relatively easy for the user to control what should execute and in which order things should execute.

2.3 Anybus CompactCom

The ABCC series is developed by the Swedish company HMS and is meant to be used to connect any type of industrial controller to any of the available networks included in the ABCC series [13]. This chapter will describe more in detail how the components of the ABCC module connects and communicate with the host application.

2.3.1 Fieldbuses and Ethernets

Today, HMS offers 20 different types of fieldbuses [13], Ethernet standards and other versions in their ABCC series, e.g. CANopen, EtherCAT, Bluetooth and USB. The main objective for the ABCC is to integrate them in machines or productions and make them available for a wide number of networking standards. The ABCC modules are run by a C driver within the host application.

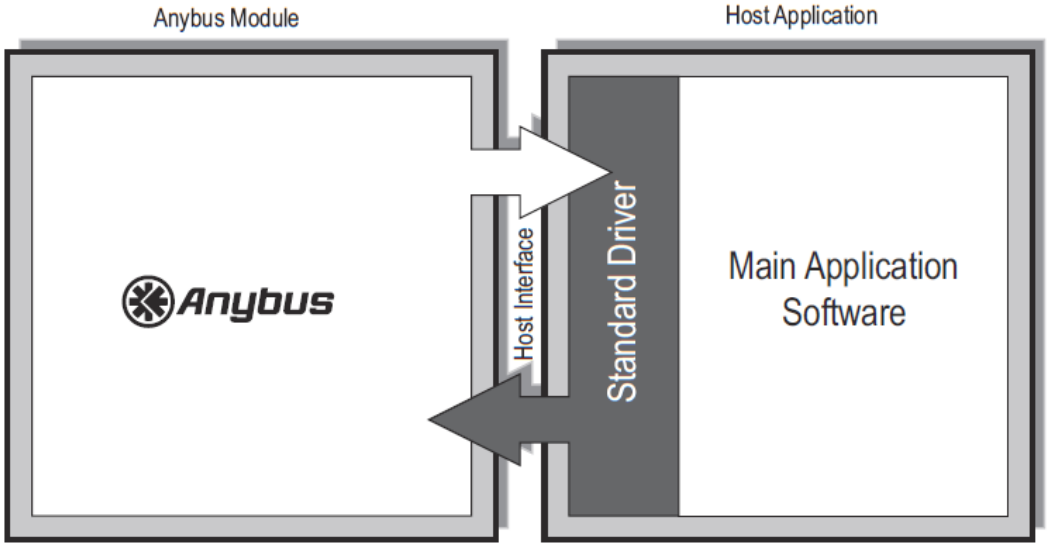


Figure 7 - Layout of the ABCC, the driver and the host application (e.g. an industrial machine) [16, p. 10]

2.3.2 Active and passive modules

The ABCC modules are divided into active and passive formats [14, p. 8]. Depending on which type of format they have they interact differently with the responding network. The passive module works as a “bridge” between the host application and the network, whereas the active module have the Anybus chip built in and translates the messages from the network to the host application and vice versa. Since the passive modules only pass on the information from the application through the module, these are not covered in this thesis.

2.3.3 Hardware design

The hardware part of the ABCC modules are not especially complicated, each has a 50 pin CompactFlash connector at the back and a network interface specific interface at the front. The network interface varies depending on which network the user wants to be able to use. Some modules also have LED indicators in the front indicating the current operations of the module.

Figure 8 - ABCC module with CANopen interface [12]

2.3.4 Communication

The communication between the host application, e.g. industrial machine, and the ABCC modules can be set up as either serial or parallel, however the passive modules only supports serial communication. Depending on what the user decides, the handling of the communication differs. Both share the same basic protocol, telegram exchange. [14, p. 8].

2.3.4.1 Telegram

The communication between the host application and the ABCC module is based on continuous telegram exchanging between the two. It is important to note that it is up to the host application to keep the communication running [15, p. 11]. It is always the host application that initiates the communication and no telegram is sent from the module unless a "start"-telegram is sent to the module. Each telegram contains of three subfields a, b and c:

a) Handshake register field

The first part of a telegram is the "Handshake register field" which is always one byte. Depending on whether the host application or the module sends the telegram the field has different functionality. If the host application sends a telegram to the ABCC module, this field controls the communication and is called Control Register. When a telegram is received from the module this field holds the current status and is called Status Register.

A more detailed insight into the control register reveals that only the four most significant bits are used:

Table 1 - Explanation of the control register [15, p. 12]

b7 (MSB)	b6	b5	b4	b3	b2	b1	b0 (LSB)
CTRL_T	CTRL_M	CTRL_R	CTRL_AUX	-	-	-	-

- CTRL_T is toggled between 1 and 0 in every message that the host application sends.
- CTRL_M is 1 if the message subfield in the current telegram is valid.
- CTRL_R is 1 if the host application is ready to receive a new command.
- CTRL_AUX is by default not used, however it can be used to indicate if the process data in the current telegram has changed from the previous telegram sent.

Example:
The host application is to send the first start telegram;
The Control register should look like:
A0h – CTRL_T and CTRL_R bit set to 1. The response from the module will then also have a high bit at STAT_T indicating that the module is in the right phase. Next message after the response has been given should look like:

20h – CTRL_T set to low and CTRL_R still high since the host application is ready to receive a command. The response from the module will also have set the STAT_T bit low, but keeping the STAT_R bit high.

In case of the status register, in telegrams received from the module, all of the eight bits are used to indicate the status of the module:

Table 2 - Explanation of the status register [15, p.12]

b7 (MSB)	b6	b5	b4	b3	b2	b1	b0 (LSB)
STAT_T	STAT_M	STAT_R	STAT_AUX	SUP	S2	S1	S0

- STAT_T will be set to the same value as the value of CTRL_T had in the last telegram received.
- STAT_M is set to 1 if the message subfield in the current telegram is valid.
- STAT_R is set to 1 if the module is ready to receive new commands.
- STAT_AUX is by default not used, same functionality as CTRL_AUX.
- SUP is used to indicate if the module is supervised by another network device.
- S[2..0] is used to indicate in which Anybus state the module is in:

S2	**S1**	**S0**	**Anybus state**
0	0	0	SETUP
0	0	1	NW_INIT
0	1	0	WAIT_PROCESS
0	1	1	IDLE
1	0	0	PROCESS_ACTIVE
1	0	1	ERROR
1	1	0	(reserved)

b) Message subfield

Next part of a message sent to or from the ABCC module is the Message Subfield. This field holds all the information regarding the object that is being addressed in the module or host application as well as the message that is being sent. The message subfield is at least 16 bytes, it can be more when using the parallel interface but on serial it is always 16 bytes.

Table 3 - Description of the message subfield [15, p. 26]

	Contents								
Offset	**b7**	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**	**b0**	**Description**
0	Source ID								ID used for keeping track of messages
1	Object								These three fields are used for specifying the destination for the message
2	Instance (lsb)								
3	Instance (msb)								
4	E								Value: Meaning 0: Message is either a command, or successful response 1: Message is an error response Value: Meaning 0: Message is a response 1: Message is a command Command, e.g. “get” or “set”.
		C							
			Command code						
5	Message data size								Specifies the size of the message. (max 255 bytes)
6	CmdExt[0]								Specifies the attribute that is being addressed.
7	CmdExt[1]								
8…n	MsgData[0…n]								Message data field

Example:

Host application	**Response from module**
A000 0000 0000 0000 0000 0000 0000 0000 00	A000 0000 0000 0000 0000 0000 0000 0000 00
40**01 0101 0041 0001 0000 0000 0000 0000 00**	2000 0000 0000 0000 0000 0000 0000 0000 00
A000 0000 0000 0000 0000 0000 0000 0000 00	E0**01 0101 0001 0201 0002 0400 0000 0000 0**0

In this example the host application first initiates the communication by sending a telegram with the CTRL_T-bit set to high, then it is requesting the attribute one from the object one (Module type). The greyed number “41” states that this is a get-request.

As visible in the response from the module, the module type is: 0402h (MSB first) which means it is an ABCC Drive Profile module.

c) Process Data Subfield

The Process Data Subfield or PDS is used when the user wants to exchange data cyclically on the network, as shown in Figure 8. The exchange of cyclic data can be done either by read process data received from the network, or write process data sent to the network [15, p. 14]. Depending on the network that is used, the representation of process data varies. It is up to the user to define the size and structure of the PDS in the host application to control the process data mapping. If the user chooses not to exchange any data cyclically the PDS will not exist in the telegrams. It is also possible to set up either read- or write process data, both doesn’t have to exist.

2.3.4.2 Serial

When the host application uses the serial interface to communicate with the module, the telegrams are a two way communication, meaning that they can both be received by the module and be sent to the host application [15, p. 17]. The settings for the serial interface are:

- Data bits: 8
- Parity: None
- Stop bits: 1

Each telegram is divided into fields as explained before, and also a CRC checksum is added to each telegram. The layout of a telegram when sent over serial interface will therefore be:

Table 4 - Layout of telegrams when sent over serial interface [15, p. 12].

1 byte	16 bytes	Up to 256 bytes	2 bytes
Handshake register field	Message subfield	Process data subfield	CRC16

(1^{st} byte) (last byte)

The CRC16 is a 16 bit Cyclic Redundancy Check which covers the whole telegram except the CRC itself [15, p. 12].

As described before the message subfield when using the serial interface, is always limited to 16 bytes, if the module or the host application has longer messages than that to send these are sent as multiple small fragments. It is up to the receiving end to bundle these small fragments and interpret the whole data as one message.

2.3.4.3 Parallel

In this thesis the parallel communication was never used when communicating with the ABCC module and therefore this is just a short explanation of parallel communication towards the ABCC.

When using the parallel interface to communicate with the module telegrams are read or written to a shared memory area [15, p. 21]. When the host application sends a new message, the application updates the Control register in the memory map. The same concept is used for the module which updates the status register to indicate that a new message is available. Because of this system, the process data and message subfield needs to be written to before accessing the control register.

a) Memory map

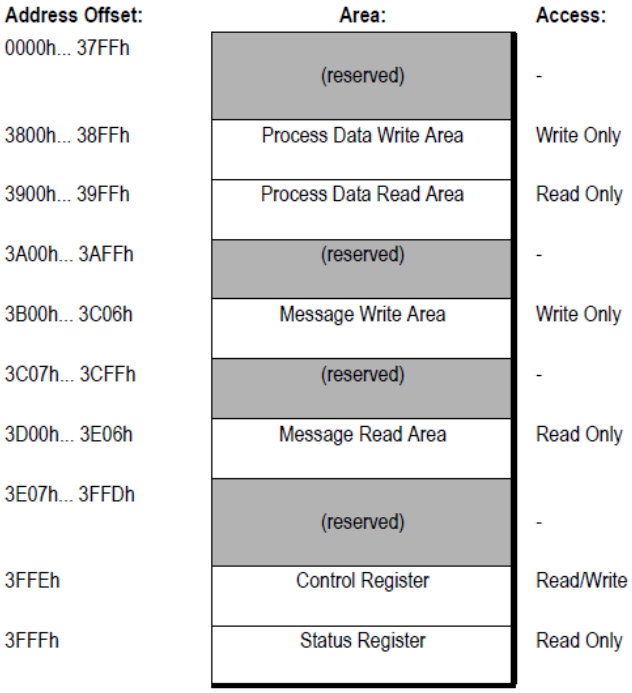


Figure 9 - Memory map for parallel interface [15, p. 21]

b) Telegram handling

When the module exchanges a new telegram this is signaled by the STAT_T-bit in the status register. To read this bit, the host application can either poll the status registers cyclically or rely on interrupt operation which is highly recommended by HMS [15, p. 22].

2.3.5 The Anybus state machine

The active modules in the ABCC series are configured to use the Anybus state machine. This means that at any given time the state machine reflects the status of the module and the network [15, p. 23].

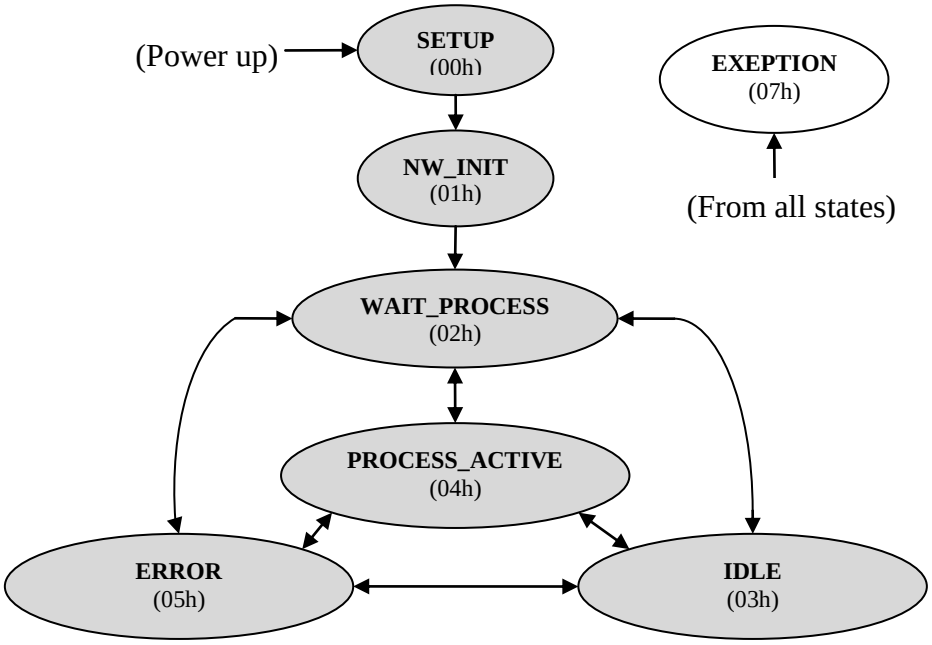


Figure 10 - Anybus state machine [15, p. 23]

2.3.5.1 States

In each state, the host application is expected to perform certain tasks, i.e. in NW_INIT the module preforms network-related initialization tasks.

When the module is configured and connected to the network it will be in the PROCESS_ACTIVE state, which means it is able to fully communicate with the network. Depending on the user action the module can transition between different states, e.g. if the network connection to the ABCC module is taken down when the module is in PROCESS_ACTIVE state, the module will switch into IDLE-state, ready to be activated again as soon as the network connection is reestablished.

2.3.6 The object model

The ABCC modules are structured in an object oriented manner, which means that in order to update or request an attribute, e.g. node address, the host application has to specify the instance and attribute that it want to update or request within a specific object. It is important to note that both the host application and the module can and will issue both request and commands towards each other, and they must both be able to respond to each request in a correct way. This is taken care of the Anybus driver easily since it is also structured in an object orientated manner. Each message is tagged with an object number, instance and attribute specifying the location of the data or setting associated with the message. There are several different objects within the ABCC module, as well as there are several objects needed to be implemented in the host application, which is described in the two coming sections.

2.3.6.1 ABCC module objects

These are objects that are implemented in the module and holds information that can be accessed or updated by the host application [15, p. 38].

a) Anybus object (01h)

This object holds data about the module that is being used. The data can be used in the host application and is i.e. firmware version and serial number. Most of the attributes within this object is of "get-type", accessible from the host application, but this objects also holds "set-parameters", e.g. setup complete, which is set by the host application when the initiation of the application is complete [15, p. 39].

b) Diagnostic object (02h)

The ABCC holds a feature for host-related diagnostics. If such an event occurs, the host application can create an instance within the Diagnostic object and when the event has been solved, the instance is simply removed by the host [15, p. 44]. This object was not covered in this thesis.

c) Network object (03h)

The network object holds information about the network. Furthermore it is used when the user wants to set up Application Data Instances (ADI's) to cyclic exchange parameters [15, p. 47].

d) Network configuration object (04h)

Within the network configuration object the parameters such as baud rate and node address are stored [15, p. 51].

2.3.6.2 Host application object

These are object that the host application has to, or in some cases should, implement to be able to communicate with the module.

a) Application data object (FEh) (Mandatory)

Within this object all of the ADI's that the user has mapped are stored. This means that when the user/module are accessing any of these ADI's the module translate such request to this object. It handles both the request to the process data buffer as well as the acyclic request from the network [15, p. 55].

b) Application object (FFh) (Optional)

Within this object there are general settings for the host application. For example if implemented it is able to support multiple languages and reset request made from the network [15, p. 64]. This object was not implemented during this thesis.

c) Network specific object (Optional)

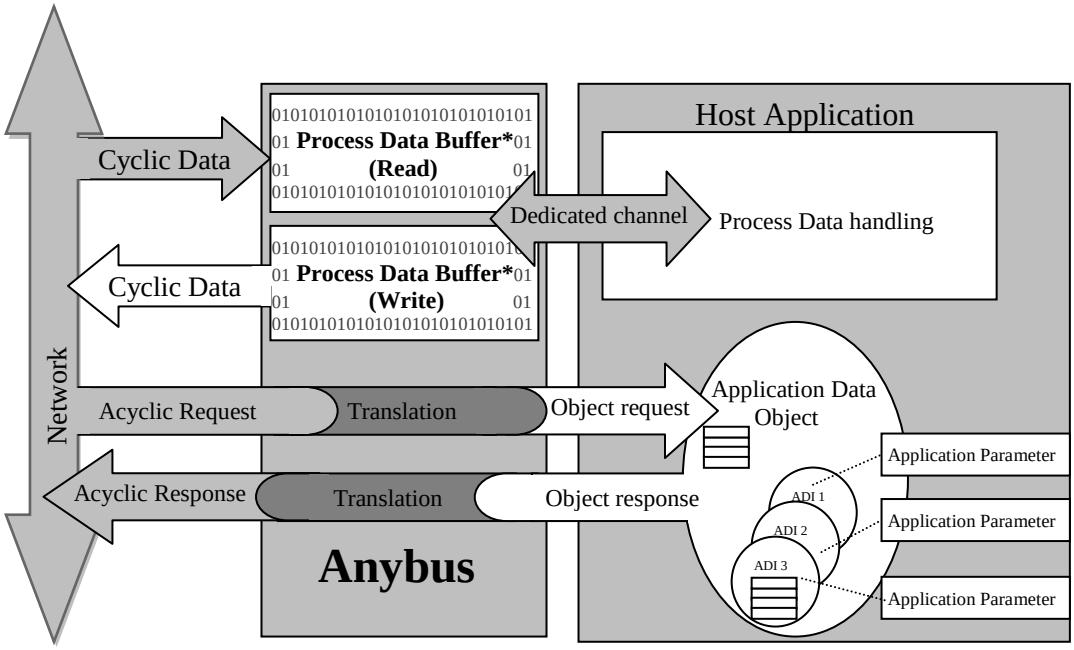
This object holds the network specific settings in the host application. The module will attempt to acquire the values within this object during startup, and if none are implemented the module will use the default value for these parameters (Although an error responses is needed to be sent to the module even if not implemented).

For example on CANopen this object refers to the CANopen object (FBh) and on CC-Link this object refers to the CC-Link host object (F7h)

2.3.7 Application Data Instances

Application Data Instances or ADI's, are parameters that are accessible by the network. There are two different ways of creating ADI's. It can either be implemented in the modules process

data buffer (as PDS) or accessible by acyclic data requests. Depending on what type that is created, different approaches are used.



*) These buffers holds data from ADI's that are mapped to process data

Figure 11 - Layout of parameters handling in ABCC module [15, p. 8]

As the picture above explains the ABCC module holds a process data buffer, readable and writeable that is accessed through a dedicated channel. The dedicated channel is a part of the telegram sent to and from the module, the "process subfield" which was explained before. It can also handle acyclic request by sending them straight through the module and to the host application. All of the parameters that are being accessible, either through the process data buffer or application data object need to be setup in the host application before the program starts. If they are to be used in the process data buffer they also need to be mapped in it as well.

ADI's are objects holding several parameters all having the same layout with fields as name, data type, number of elements and value etc.

2.3.7.1 ADI's in process data buffer

The ADI's that are mapped in the process data buffer need to be accessible to both the module and the host application. This is done by the process data subfield that is used in the telegrams that are exchanged between the ABCC module and the host application. If an ADI is mapped to be read process data the value of that ADI is present in the process data subfield of the telegram that is sent from the module. If the ADI is mapped to be write process data, the value of that ADI is present in the process data of the telegram sent from the host application.

The process data buffer is represented in different ways on the network depending on which type of network that is used. For instance on CANopen it is represented as PDO's and on DeviceNet they are represented as dedicated CIP objects. It is up to the user to make sure that the handling of these objects on the network side is handled in the right manner [15, p. 14].

2.3.7.2 Acyclic requested ADI

ADI's that are to be exchanged in acyclic requests on the network are stored in the host application. When a network acyclic request of a certain acyclic ADI's value is sent that

message is sent through the module to the host application, as seen in Figure 8. It is then up to the host application to provide a correct response to that request. For the host application to be able to know which ADI's that is requested the module translates the request into the corresponding instance in the Application data object (FEh) [15, p. 55].

As with the representation of the ADI's in the process data buffer, acyclic ADI's are also represented in different ways depending on which network that is used. On CANopen for example acyclic ADI's are represented as SDO objects.

Chapter 3
Approach

This chapter explains how the practical part of the thesis was carried out and the steps that where taken during the thesis.

3.1 Starter kit

In order to be able to communicate with the ABCC module, a computer had to be connected to the module; therefor WireFlow purchased a starter kit from HMS, which is intended to help developers learn the basics of ABCC and speed up the implementation of ABCC module in host applications.

The starter kit consists of a PCB with a CompactFlash connector (A), for connecting the ABCC module, and a serial COM-port (D) for communicating with the ABCC module. The board also holds a power socket (E), a reset switch (C) and a baud rate switch (B).

With the starter kit HMS there was also a full driver provided to be used in the host application, and also a demo application, called Anybus SDK, with some settings already in place to be able to set up a small "industrial network environment". This SDK is written in C# and is run by using the command prompt on a PC.

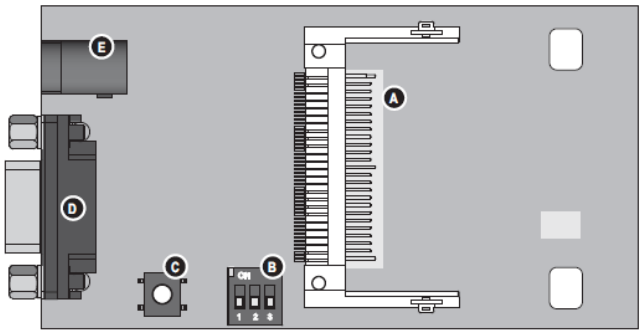


Figure 12 - Starter kit board [11]

3.2 LabVIEW communication

The main thing for this thesis was to get the ABCC module communicating via NI LabVIEW towards the CompactRIO. This was done by creating a demo application, or SDK (similar to the Anybus SDK), and test the communication.

A LabVIEW environment was set up with a serial COM-connector and an input text field for sending telegrams to the ABCC module. A VI was also created for adding the CRC field automatically to the telegram that was sent to the module.

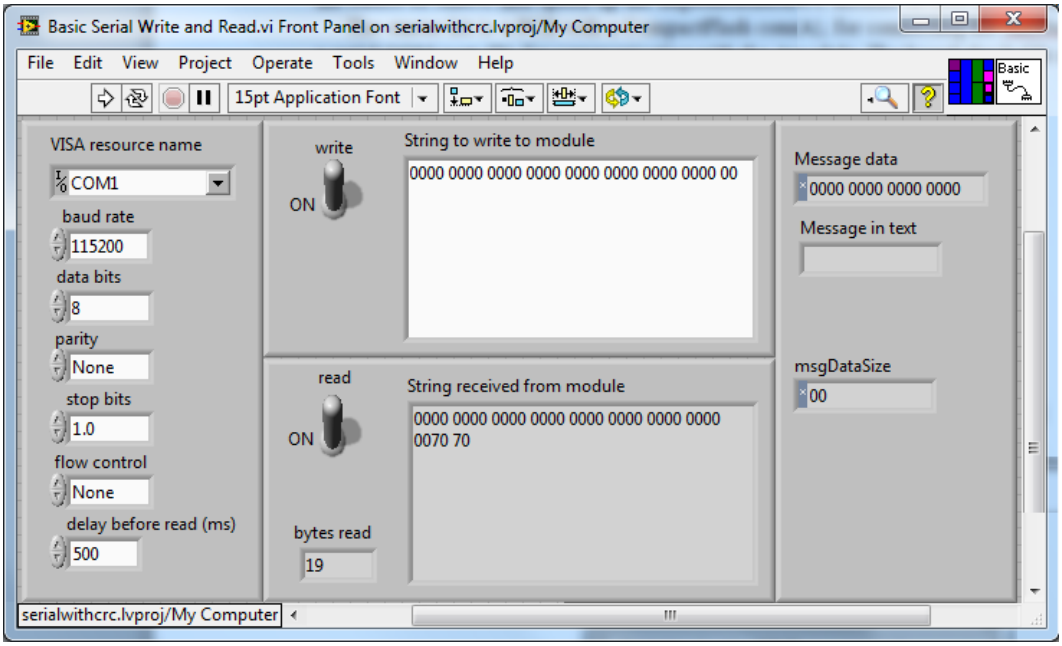


Figure 13 - LabVIEW serial interface

3.3 Modules

The ABCC module that was used during this thesis was an ABCC module with the fieldbus CANopen. With the CANopen module a network was set up consisting of: Starter kit connected to the computer via a serial COM-port and the module connected to a NI PXI chassis, with a CAN port.

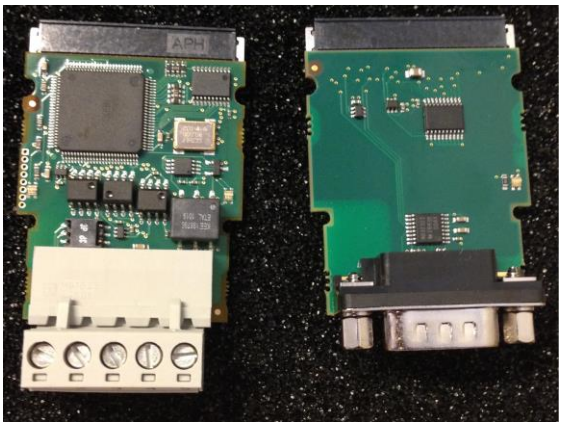


Figure 14 - CC-Link and RS-232 module without housing

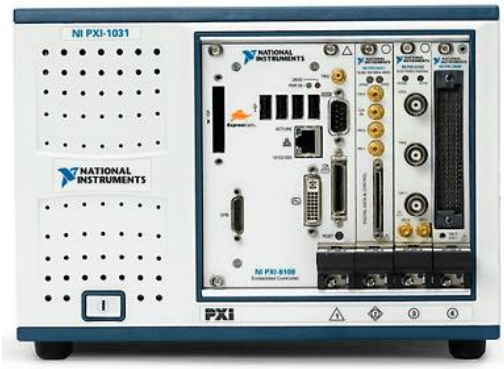


Figure 15 - NI PXI 1031 Chassis [21]

3.4 CANopen

CANopen is a network standard most commonly used in industrial networks. The protocol for CANopen is based on the CAN protocol but is on a higher level, making it easier to work with [7]. LabVIEW also holds several example libraries for communicating over CANopen, that where really helpful during the thesis.

3.5 Full testing of Anybus SDK

With the different components in place and a complete environment for communication from the PC to the CANopen network and vice versa, a test was run in order to get a wider knowledge of how the Anybus SDK works. In order to solve this, a serial sniffer was created which was able to log all the communication be sent and received to and from the PC and the starter kit. To record the serial communication a LabVIEW VI was created that saved all the data to a file and stored it on the PC.

The LabVIEW code for communication through CANopen is quite simple to setup, although to be able to access the right parameters that are set up in the ABCC module by the host application (the Anybus SDK i.e.) some extensive studying of the CANopen protocol had to be made.

3.5.1 Anybus SDK

In the Anybus SDK that comes with the starter kit, the host application is set up to be a fictional motor with five accessible parameters available from the network, ADI's. Three of these are set up as PDO and two as acyclic data requests. The five parameters are:

1. **Actual speed,** a parameter mapped as input process data, readable from network. (Read PDO)
2. **Temperature,** a parameter that lets the user to set this value from within the application and can be read from network using acyclic data read requests. (Acyclic data read requests)
3. **Ref speed,** a parameter that is mapped as output process data and can be written by the fieldbus master. (Write PDO)
4. **Direction,** a parameter mapped as output process data and can be written by the fieldbus master. (Write PDO)
5. **Poles,** a parameter that the user can set from the network using acyclic data write requests. (Acyclic data write requests)

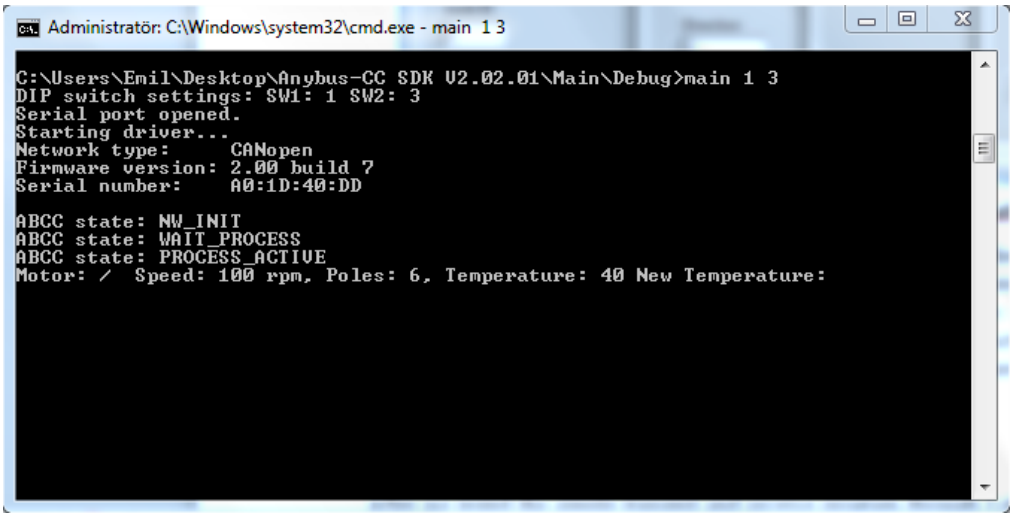


Figure 16 - Command prompt of Anybus SDK

3.5.2 CANopen communication to Anybus SDK

To be able to fully understand the Anybus SDK a small industrial network was set up, as explained before. The network communication towards the Anybus SDK would have to be CANopen and the programing environment used was LabVIEW, since that is the only way of

communicating through the PXI chassis. In LabVIEW there are libraries available for easy usage and setup of external controllers, CAN controllers or serial controllers etc. The CANopen library and the examples that came with it were used to setup an easy environment for communicating with the ABCC module.

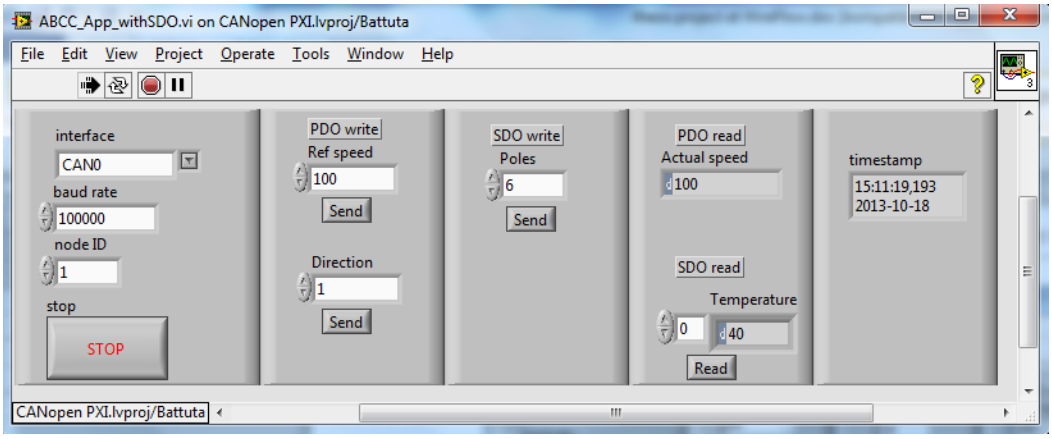


Figure 17 - CANopen interface in LabVIEW

In this application there are interface controllers on the left, and controllers to access the writeable parameters in the process data buffer, called PDO's. There is also a control for changing the parameter "Poles" that is accessible through acyclic data write request (SDO). Furthermore there is one indicator for the readable parameter in the process data buffer "Actual speed" and an indicator for the parameter accessible through acyclic data read request "Temperature". The CANopen fieldbus is structured, like the ABCC modules, with object orientation and in order to set up these parameters to access the right objects studying of the CANopen protocol was needed. [6]

3.5.3 Conclusion of tests towards Anybus SDK

By studying the telegrams exchanged between the ABCC module and the host application it was established how the messaging between the two works. It was also demonstrated that the module handle both read and write requests through the process data buffer and the acyclic requests. It was established that HMS has setup their ADI's and how they are mapped in the process data buffer. Although all of this could have been learned by reading and studying the C# code for the Anybus SDK this saved a lot of time and also gave a good understanding of the Anybus driver. But this was just the start, since the goal for WireFlow is to implement the ABCC module in their cRIO module it needed to be communicating through another programming language, LabVIEW.

3.6 LabVIEW SDK

The next goal, after testing the Anybus SDK against the CANopen interface, was to develop the Anybus SDK in LabVIEW, translating both the driver of the Anybus and the host application. The host application in this case is the part of the application that handles the user input and updates the needed parameters according to what the user inputs. Since the LabVIEW SDK would only be able to be tested towards the CANopen module from Anybus, a decision was made to "hardcode" some of the functionality that is specific to just that module. Even though it is expected to be working with several different modules, since most of the commands are common among all of the ABCC series modules.

The developing process was started by testing some simple transmit and receive commands to the ABCC module through a pre-defined program in LabVIEW that uses a

serial port for communication. In the beginning the commands sent was just the exact ones that was “sniffed” earlier in testing and all the commands gave valid responses. Then step by step the building of the SDK started in LabVIEW, making it more and more automatic. To control what message where sent to the module the serial sniffer was used again to verify that all commands where given valid responses. The Anybus state machine that the ABCC modules use was replicated and was the basis for the LabVIEW SDK. (See picture below)

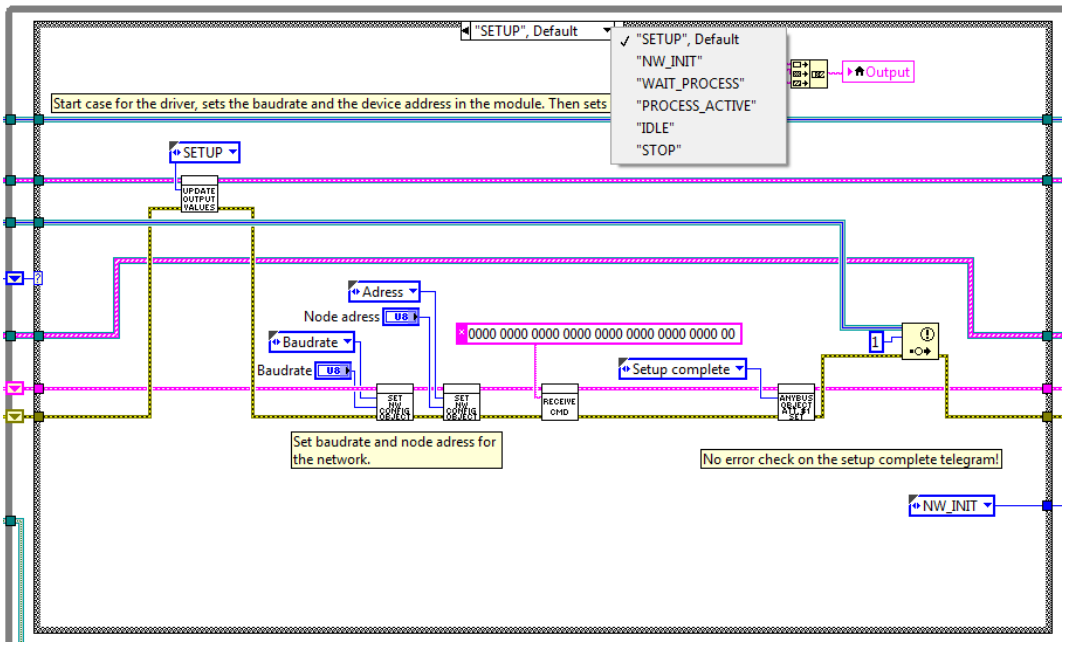


Figure 18 - Picture over state machine in LabVIEW SDK

The next step was to create functions for handling the commands that the module sends during NW_INIT-phase, containing some of the “hardcoded” data as explained before. After implementing all the needed parts of the driver and testing it, the SDK was up and running. The SDK was then tested towards the CANopen interface using the same network setup as earlier. This proved to be working very well, although some minor modifications were needed since the mistake of not separating the driver of the SDK and the application was made, this was fixed in an updated version of the SDK. Once the SDK was completed it was able to run all commands as it was with the ABCC SDK.

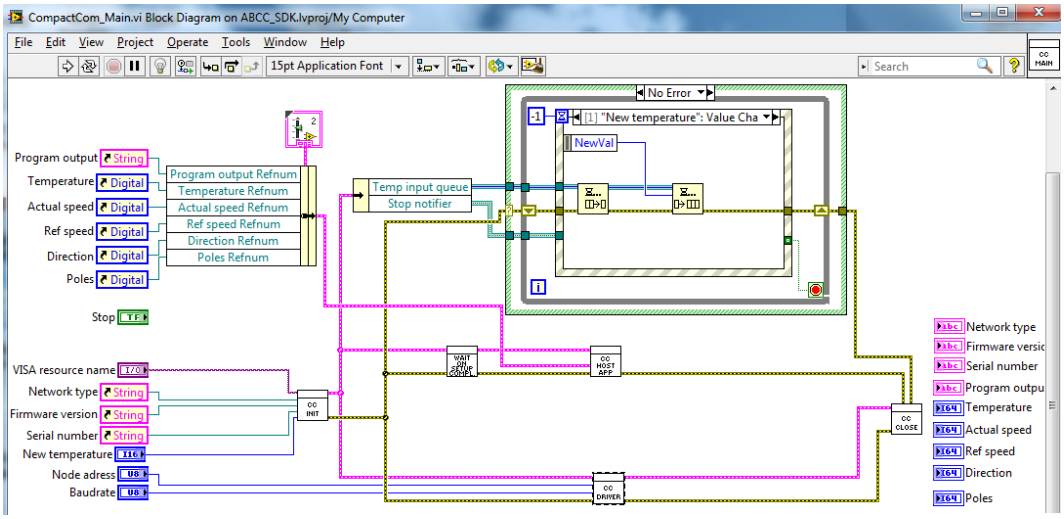


Figure 19 - Main application code, block diagram

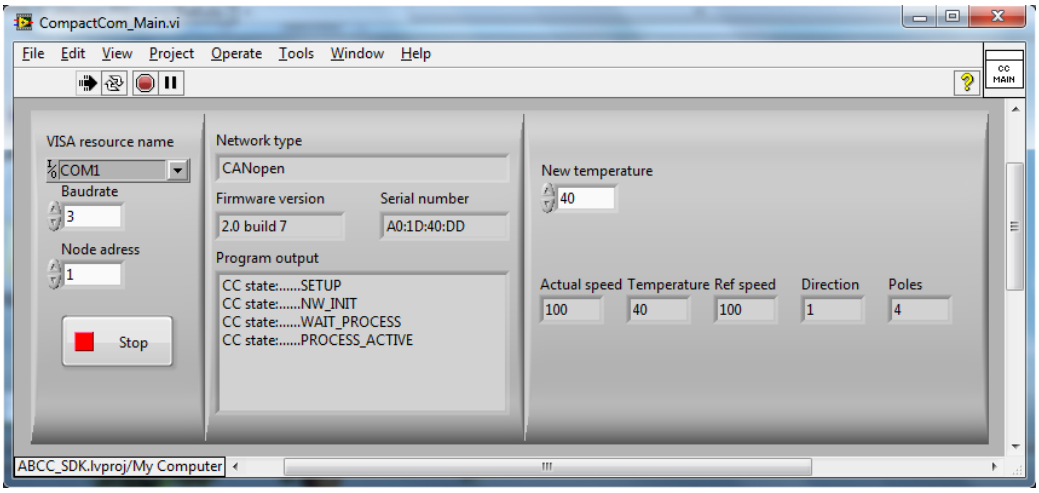


Figure 20 – Main application (SDK) GUI in LabVIEW

3.6.1 Conclusion of the LabVIEW SDK

In developing the LabVIEW SDK the recorded telegrams from the serial sniffer was a great help. With the telegrams from the Anybus SDK to compare and work against the needed functionality and implementations became clear. Although the SDK only supports the CANopen interface it is hopefully something that WireFlow can use as a guide in future work with his product.

3.7 LabVIEW SDK and ABCC module in cRIO chassis

The finished LabVIEW SDK meant that a fully functional program is able to run together with a cRIO chassis. The next step of the developing process was to make a prototype of a cRIO module that would integrate the ABCC module. The first intent was to use a ready module for a cRIO chassis called Tangent Blue. The Tangent Blue module has all the pins that were needed to connect it to the ABCC module. In order to be able to run the Tangent Blue module in the cRIO chassis with the module an FPGA driver was needed to control it. Since the developing of such a driver requires quite a lot of experience, WireFlow helped with that driver. While WireFlow worked on the driver the decision was made to make a “mockup” and try to answer the question on whether all the components would fit inside a cRIO module. With the finished mockup it was clear that the ABCC module would fit inside the cRIO module. The next step then was decided to be to design a PCB just for the module that would provide a complete prototype. The Tangent Blue integration was therefore scrapped. When the designing of a PCB started helpful guides and tips was provided by WireFlow. The layout of the PCB was created by reusing a layout of an old module that WireFlow have developed before. The process started with the creating of a schematic with all the components that were needed. The PCB design software that was used was DesignSpark which is a free software that WireFlow have used before. DesignSpark is an easy software to start working with but when creating a pin connector for the CompactFlash connector it became quite challenging. When the schematic was finished it was translated in to a PCB using a built-in function. Then the components were placed on the PCB and connected with wires. In the design a two layer PCB was used with components on both sides in order to fit them all.

3.7.1 PCB design notes

In order to be able to acquire a copy of the finished PCB in time only the vital components, to get the module to work was considered. Some shortcuts were taken;

- The possible reuse of the MOSI (Master data Output, Slave data Input) and MSIO (Master data Input, Slave data output) pins on the DSUB connector was not implemented. Instead two other DIO (Digital IO) pins were used to set up the serial connection. In the future WireFlow would want to reuse the MOSI and MISO pins since that would work as well, although more components need to be added.
- An investigation of how to implement the SLEEP pin (which all modules use to save power) in the cRIO module was done and found that the ABCC module draws approx. 50 % less power when the signal to the Reset-pin is active. (It's active when low)
- The consideration was also made that WireFlow will need to add more power externally to the ABCC module, if required. Adding such a connector at the front of the module would require space on the PCB which was left out.
- Since this was just a prototype a DIP-switch was added to be able to control the baud rate of the serial port interface. This should be done automatically instead in future work.
- Another part to take into consideration when designing the PCB for this module is the LED diodes that the ABCC module uses. When the ABCC module sits in the plastic cover, these are directed through the front of the module using plastic bracket. WireFlow could reuse this feature or come up with a simpler connection for these.
- The GND on the prototype is shared between the cRIO module and the ABCC module, this is something that should be avoided and WireFlow should use another power regulator with different ground pins for each in-/output. The other IO pins (Rx, Tx and Reset) should also be included in this electrical isolation.

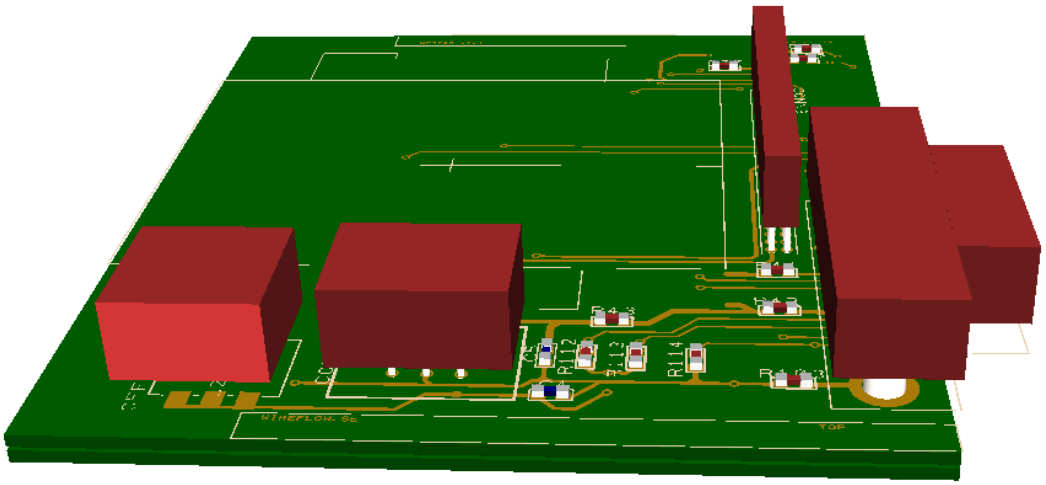


Figure 21 – 3D view of the PCB for the module

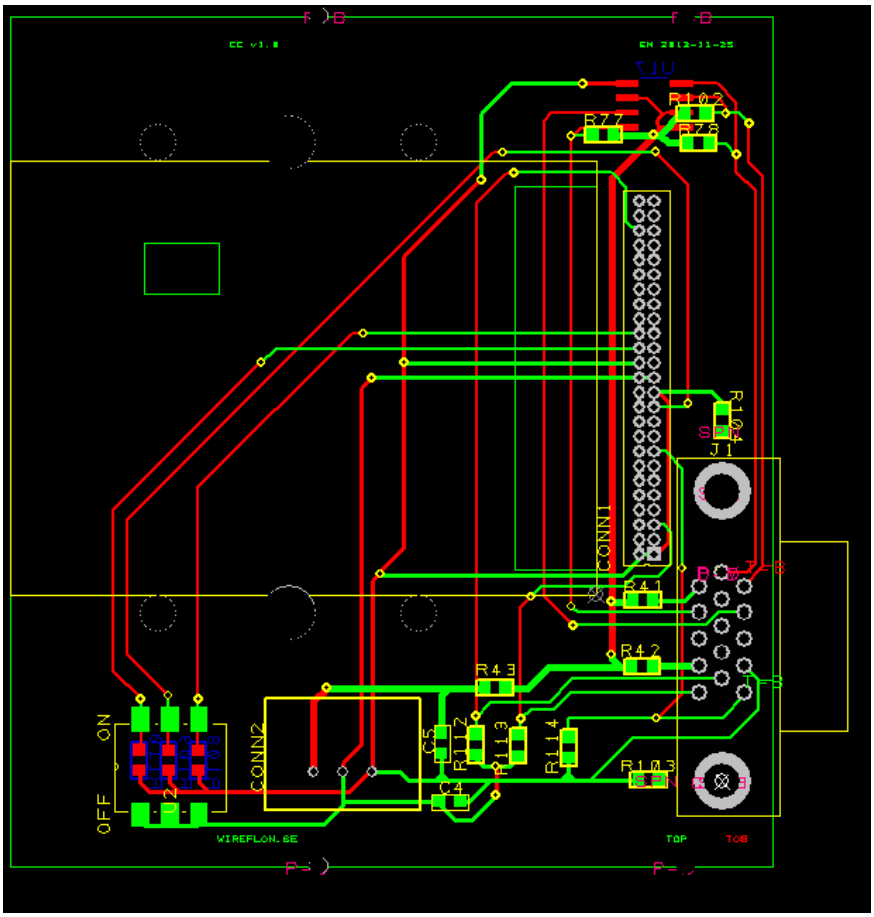


Figure 22 - Complete PCB layout

3.7.2 Fully functional prototype

With all the needed components ordered and delivered, the assembly of the prototype began. There were some minor changes made (some things got mirrored on the PCB during development) to be done to the PCB. In the end everything fit well and the whole prototype looked good.

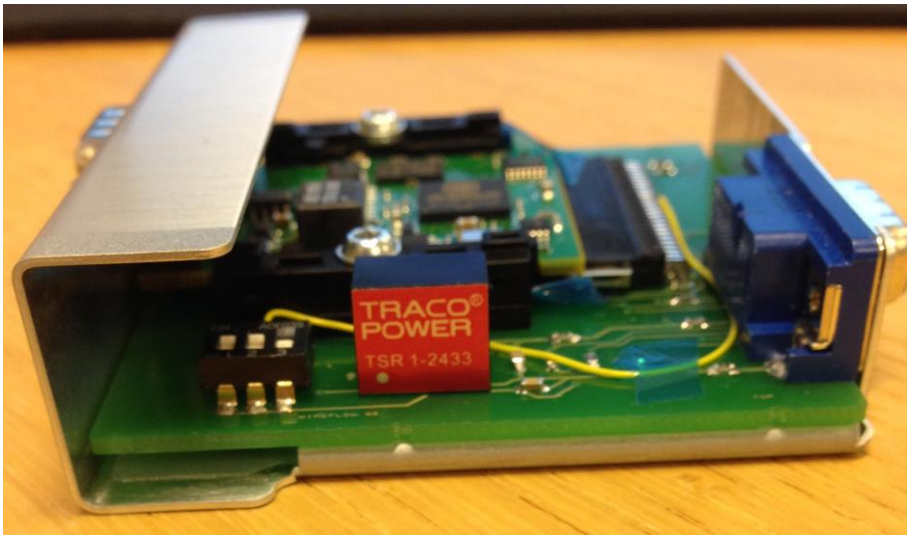


Figure 23 - Complete assembled prototype without the back panel

After assembling the prototype some measurements were made to check that all power levels were within the limits and also made sure that all pins were connected to the tight paths. Everything looked good and the next step was to test the module in a cRIO chassis. It worked as good as it had done before in the starter kit environment and in the end fully functional prototype was complete.

Chapter 4
Evaluation

The goal with this thesis was to answer four questions;

- Will there be enough space in the cRIO module for the ABCC module, including extra needed electronics?
- Will the ABCC module meet the power requirements for cRIO modules?
- How will the software interfaces between ABCC module and cRIO work together?
- Are there any similar products available in the market today? What fieldbuses doesn't exist for cRIO modules today?

In this chapter, we will give answers to these questions.

4.1 Requirements

When developing a module for the cRIO standard, certain requirements are set on the module and the components within the module. Additionally the ABCC modules require certain power and voltage to be able to run as intended. In this part the requirements will be explained and will be evaluated whether or not the ABCC module will fit inside of the cRIO module, both in terms of physical size and in terms of requirements.

4.1.1 cRIO modules

For cRIO modules certain recommendation and rules are available for developers. These are thoroughly described in the CompactRIO Module Developer Kit User Manual - MDK and here these are summarized.

4.1.1.1 Dimension

The PCB that is fitted inside the cRIO module must have an outer dimension of: 73.38 x 66.04 x 1.57 mm [20, p. 13]. The components fitted inside the module cannot be higher than 13.46 mm if fitted in the main area of the PCB.

The I/O connector fitted to the front panel cannot be wider than 73.38 mm and cannot be higher than 13.46 mm on the primary side (top) and 2.64 mm high on the secondary side (bottom). Same dimensions as for the components [20, p.20].

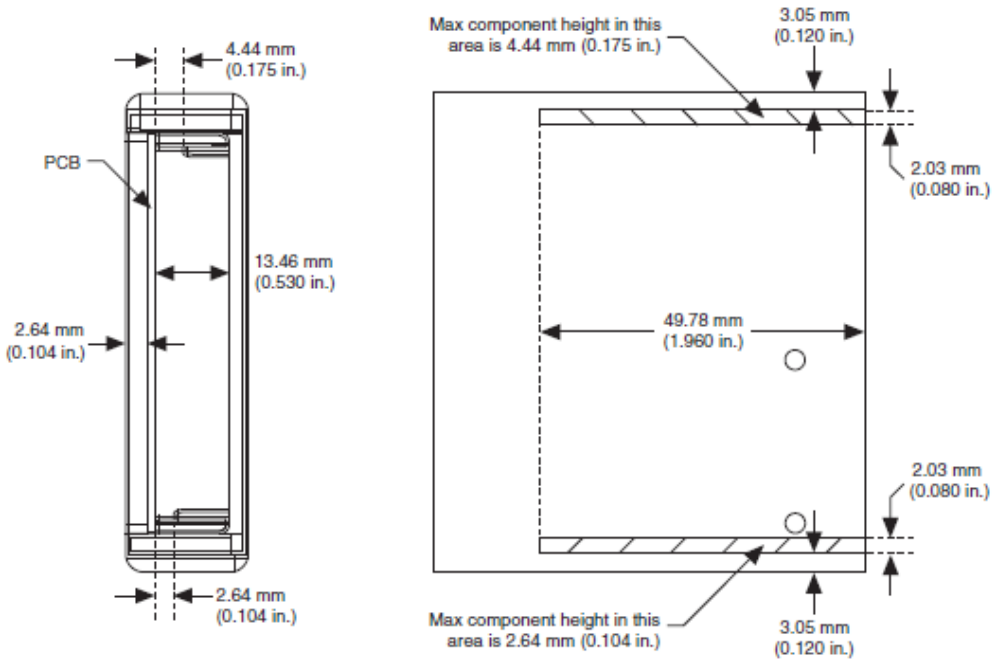


Figure 24 - cRIO height restrictions [20, p. 19]

4.1.1.2 Current consumption

It is stated in the CompactRIO MDK p. 124 [20], that the peak of the current draw from the power line shall remain within 0 mA and 200mA when developing a cRIO module.

4.1.1.3 Thermal requirements

The thermal requirement for modules is that they shall be able to operate within -40 – 70°C. NI has also listed that modules should be able to be stored at temperatures between -40 – 85°C. [20, p. 131]. Another rule that is detailed in the cRIO MDK [20] is that the modules have certain power limits, depending on the type of components that are used in them, this power limit is constant through the temperature range described before (-40 to 70°C):

- 0.625 W for modules with 85°C rated components
- 1.5 W for modules with 100°C rated components

This means for example that if a component that is used in the cRIO is rated at 85°C, then the module cannot draw more than 0.625 W at any time.

4.1.2 ABCC modules

The ABCC modules have certain requirements that need to be fulfilled in order for them to work.

4.1.2.1 Dimensions

When removing the housing from the ABCC module it is quite small, in fact, the module then contains of a PCB board with outer dimensions of: 50.5 x 36.6 x 1.6 mm and the fieldbus connector at the front. The fieldbus connectors vary in size from fieldbus to fieldbus;

Height from ABCC PCB to top of connector, width is also included [10, p. 9-12]:

- D-SUB: 12.6 x 30.8 mm
- RJ45: 9.5 x 15.5 mm
- RJ45 (2-Port): 12.3 x 33.6 mm
- USB: 10.9 x 12 mm

- Pluggable Screw Terminal: 8.6 x 27.4 mm
- BNC, 2-Port: 11 x 28.2 mm
- CompoNet: 10 x 33.3 mm

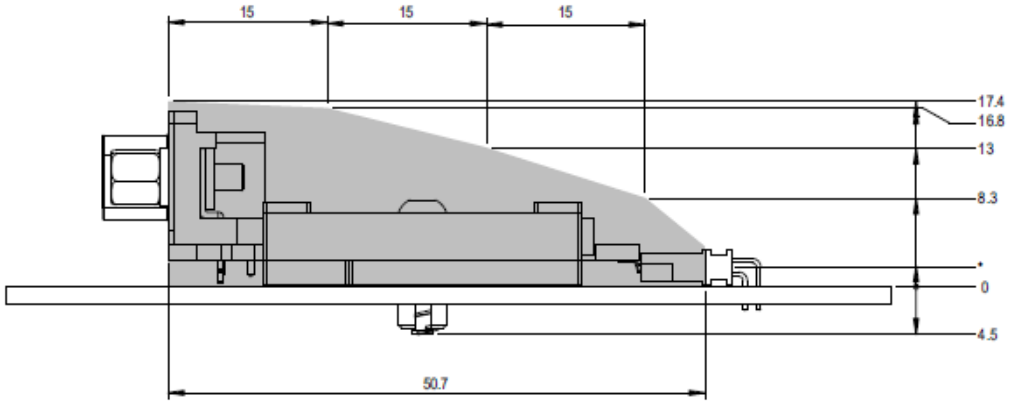


Figure 25 - Measurements of ABCC module with housing [10, p. 6]

4.1.2.2 Current consumption

In every network appendix for the ABCC modules HMS has written the current consumption for that type of module.

Table 5 - The supported networking systems and their actual power consumption [13].

#	Network	Consumption (mA)	Power (W) P=V*I	Connector type
1	BACnet/IP 2-Port	380	1,254	RJ45 (2-port)
2	BACnet MSTP	317	1,0461	Pluggable Screw Terminal
3	Bluetooth (Passive)	50	0,165	None
4	CANopen	185	0,6105	D-SUB 9pin
5	CC-Link	280	0,924	Pluggable Screw Terminal
6	ControlNet	660	2,178	BNC
7	CompoNet	188	0,6204	CompoNet
8	DeviceNet	65	0,2145	Pluggable Screw Terminal
9	EtherCAT	370	1,221	RJ45 (2-port)
10	Ethernet/IP 1-/2-Port	200/380	0,66/1,254	RJ45 (1-/2-port)
11	Modbus RTU	210	0,693	D-SUB 9pin
12	Modbus-TCP	400	1,32	RJ45 (2-port)
13	Profibus DP-V1	230	0,759	D-SUB 9pin
14	Profinet 1-/2-Port	200/380	0,66/1,254	RJ45 (1-/2-port)
15	RS232 (Passive)	22.5	0,07425	D-SUB 9pin
16	RS422/485 (Passive)	170	0,561	D-SUB 9pin
17	Sercos III	400	1,32	RJ45 (2-port)
18	USB (Passive)	50	0,165	USB

4.1.2.3 Thermal requirements

The ABCC modules are fully operational between -40 to 85 °C [10, Appendix A] these can also withstand the same temperatures when being stored.

4.1.3 Conclusion of requirements

The next part will summarize all the above mentioned requirements and give a good evaluation to what needs to be considered when designing the real product.

4.1.3.1 Dimension

At first glance of the requirements for the cRIO module, it is clear to see that the ABCC module is to high (at least with the D-SUB connector), 16.8 mm in contrast to the height requirements for the cRIO which is 13.46 mm. The gap means that the ABCC modules with Pluggable Screw Terminal only will meet the rules in the cRIO MDK. This was also discovered when building the mockup, since a cutout had to be made in the cRIO module outer case in order to fit it completely. In other areas there is no problem in fitting the module, since the connector is the highest component on the module. For a more detailed description, see section 5.2. When choosing components to be used in the cRIO module it is also important to take into considerations the dimensions of these, such as voltage regulator etc.

4.1.3.2 Current consumptions

As stated in the CompactRIO Module Development Kit user manual, MDK, the maximum current peak that is allowed is 200mA. This however is meant as a rule when using 5 Volt for the components within the module. In this case a DC-DC converter is used to limit the voltage to 3.3 volt to power the ABCC module. This means that the current peak from the DC-DC converter is a bit higher.

*P=U*I (P=Power, U=Voltage and I=Current).*

When using 5 volt:

*5V*200mA = 1 W.*

When using 3.3 volt (with 85% efficiency):

*(1W*0,85)/3,3V = 257 mA.*

This means that in order to be able to use the ABCC modules together with the cRIO module, the ABCC module cannot draw more than 257mA of current. This implies that 11 of the 20 available ABCC modules will work within the cRIO module. However, the rule in the MDK can be taken out of consideration if an external power supply is used to power the electronics inside the module. This is something that has been discussed with WireFlow. So in the case of power consumption there is not really an issue for the modules to be integrated in the cRIO modules.

4.1.3.3 Thermal

Since the ABCC module will be installed inside the cRIO module it is important to make sure that the inside of the cRIO module meets the requirements of the ABCC modules temperature range, -40 - 85°C. With the rule stated in the cRIO MDK [20] that modules cannot exceed the power limit of 0.625 W for modules with 85°C rated components it is important for WireFlow to make sure that the power doesn't exceed this limit, otherwise the inside of the cRIO module may be too hot for the ABCC module to be fully functional. Since the current drawn of each module is known as well as the voltage level required to power them, a column with the power is shown in **Table 5.** This shows that 7 of the 20 ABCC modules will meet the thermal requirements stated in the cRIO MDK [20]. In order to support the remaining 13 WireFlow would have to work around this issue by e.g. implementing a cooling or ventilation system. It should also be noted that for 100°C rated components the maximum power limit is 1.5 W, which covers all but the ControlNet ABCC module.

4.2 How to fit a ABCC module inside a cRIO module

One of the most interesting parts of this thesis was for WireFlow to get an understanding of how to fit the ABCC module inside the cRIO module, and in order to test this properly the decision was made to build a mockup, a hardware design of the needed components. This part will give an understanding of how this was done and what an end-designer needs to think of when making this complete module.

4.2.1 cRIO module components

The cRIO module consists of two major components:

- Outer shell, divided in a front panel (aluminum, silver) and a back panel (metal, blue)
- PCB, with dimensions of 73.38 x 66.04 mm.

Figure 26 - Empty cRIO module

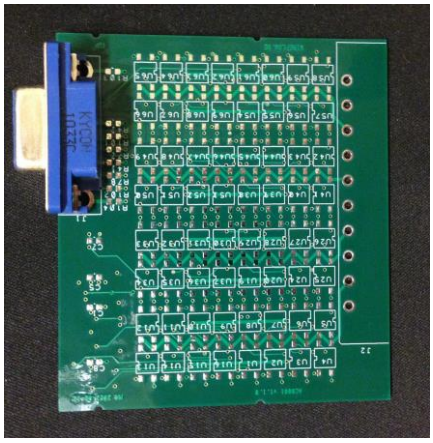


Figure 27 - PCB for cRIO module with DSUB

As visible at the PCB, a D-SUB connector has to be fitted at the back of the PCB this is used when plugging the module into a cRIO chassis.

4.2.2 ABCC module components

All ABCC modules have an outer casing which is same for all of them, the only difference between different modules are the connector and the LED at the front.

4.2.2.1 Removing outer shell

When the ABCC modules are delivered they were enclosed in a plastic housing, and enclosed in this plastic housing the ABCC module were too high to fit inside the cRIO module. The modules were therefore taken apart, something that is possible since HMS delivers modules without the casing as well.

Figure 28 - CC-Link module with housing

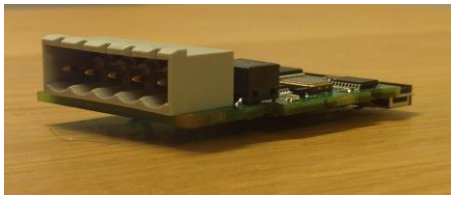


Figure 29 - CC-Link module without housing

As visible in the pictures above, the height of the module without the plastic housing is far less. The dimensions of the width are also less which gives more space on the PCB to fit other components.

4.2.3 CompactFlash connector

As mentioned previously the PCB for the cRIO module holds a DSUB connector at the back of it. This causes problems since the CompactFlash connector used for connecting to the ABCC module requires quite a lot of space. This is mainly because the connector was design to be mounted on the surface of the PCB. Because of this a decision was made to try to use a smaller connector, and a connector with through-hole pins was found which requires less space.

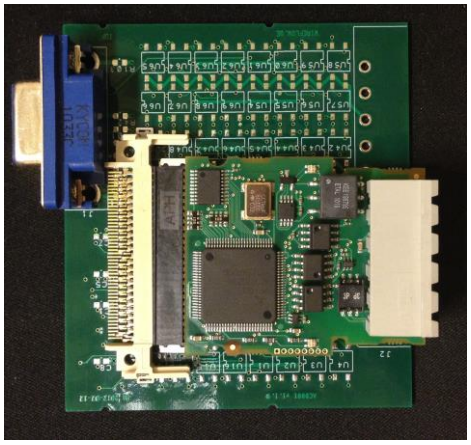


Figure 30 - PCB with CC-Link module and CompactFlash connector

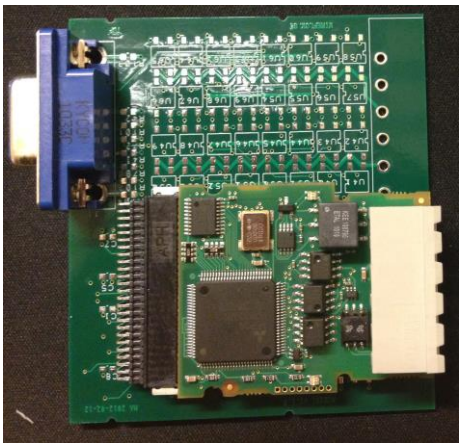


Figure 31 - PCB with CC-Link module and through-hole connector

4.2.4 Front panel of cRIO module

The front panel of the cRIO module is where the connection to the fieldbus will be accessible. At first the plan was to reuse the front panel of the ABCC module but after discussion with WireFlow it was obvious that it didn't look good enough. So instead a cutout would be made, using a drill, for the fieldbus connector in the front panel of the cRIO module.

4.2.5 Alignment of ABCC module on cRIO module PCB

The ABCC module was measured and aligned on the PCB and drilled holes for the mounting kit of the ABCC module and then cut out a horizontal hole for the slim connector to the CompactFlash connector. By doing this and creating the holes in the front panel it was proven that the ABCC module would fit inside the cRIO module.

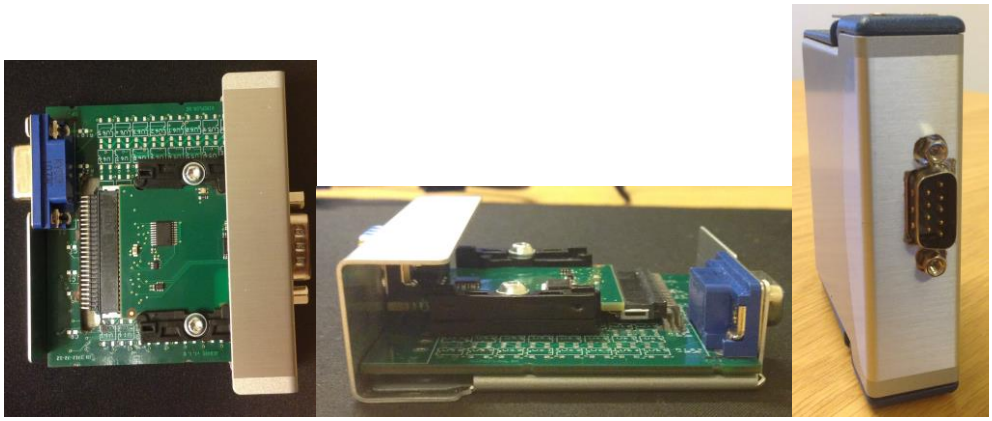


Figure 32 - Pictures showing the ABCC module inside the cRIO module

In this mockup the RS232 module from the ABCC collection was used; this was used because it has the same connector as the CANopen module, D-SUB 9pin. When the mockup was made the requirements of cRIO modules had not been examined, and it was not known that the connector was too high, although we manage to fit it inside.

4.2.6 Creating a prototype PCB

As discussed before (in LabVIEW SDK and ABCC module in cRIO chassis) the decision was made, since all the components were available, to build a real prototype. It was also decided that a PCB would be created, holding all the needed components to test the ABCC CANopen module.

4.2.7 Conclusion

The conclusion of the question on how to fit the ABCC module inside the cRIO module is that it is possible. Although a normal CompactFlash connector cannot be used since that requires too much space on the PCB. Since the mockup was made using a ABCC module with a D-SUB connector at the front (figure 29), it is known that no matter which ABCC module that will be used in future work, it will physically fit inside the module, since the D-SUB connector is the highest one (Dimensions). Although weighing in the height restrictions rules for the cRIO module, it is not possible to use such a PCB layout as was created.

Since some of the ABCC modules requires more current than it is possible to get from the cRIO chassis, an external power source need to be used. We had this in mind when designing the PCB for our prototype, although such a connector was not placed during this thesis. However there is enough space for such a connector to be placed beside the ABCC module on the PCB with the connector having its own hole cut out in the front panel of the cRIO module.

Another important detail to take into account when and if WireFlow decide to take this product into production is the placement of the ABCC module on the PCB. Since the initial point was not to make a functioning prototype, the placement of the module on the PCB is not the most optimal. There is a possibility of moving the module closer to the side of the PCB, making room for more components alongside it.

4.3 cRIO interface together with the ABCC module

This part will discuss the challenges on how to integrate the communications towards the ABCC module in LabVIEW.

4.3.1 Object hierarchy

One of the challenges with integrating the driver software in LabVIEW was the object hierarchy that exists on the ABCC C# driver. This causes problems since there are a lot of objects needed to be implemented in order for the driver to support them all. When developing the LabVIEW SDK driver this became known. However since time was limited the decision was made to only implement the support needed for just the module was going to be used. When designing a driver to support all the available modules it is best to base that the code on parameters that are "gettable" from the module at startup, such as "Module-type", "Network-type" etc.

4.3.2 Conclusion

The working LabVIEW SDK driver will hopefully give WireFlow a good sense of what they need to expand and implement and that this code can be a guide for them to follow in order to implement the needed functionality.

4.4 Market research

In order to develop a successful product the company has to know what is available on in today's market WireFlow's main concern was not to "reinvent the wheel" and develop a product that already exists. WireFlow want to be able to give NI's users an extra dimension by being able to give the users possibility to connect to any type of network standard available today. During this thesis we found that, from what networking standards HMS will provide via the Anybus system these are available and not.

Table 6 - Market analysis [18]

Anybus protocol	On market	Name & link to target (M=Master, S=Slave)	Price (SEK)
Fieldbus versions:			
BACNet MS/TP			
CANopen	Yes	NI 9881 http://sine.ni.com/nips/cds/view/p/lang/sv/nid/209998	6 090
CC-Link			
CompoNet			
ControlNet			
DeviceNet	Yes	NI 9882 http://sine.ni.com/nips/cds/view/p/lang/sv/nid/211943	6 090
Modbus RTU			
Profibus	Yes	http://sine.ni.com/nips/cds/view/p/lang/sv/nid/208386(M/S) http://sine.ni.com/nips/cds/view/p/lang/sv/nid/208387 (S)	17 230 10 090
Ethernet versions:			
BACNet/IP			
EtherCAT			
EtherNet/IP	Yes	http://www.sea-gmbh.com/en/products/compactrio-products/wireless-technology/wlan/	€869
Modbus TCP			
Powerlink	Yes	[28]	
Profinet	Yes	http://sine.ni.com/nips/cds/view/p/lang/sv/nid/211935 (S)	12 760
Sercos III			
Other versions:			
Bluetooth			
RS-232	Yes	NI 9870 http://sine.ni.com/nips/cds/view/p/lang/sv/nid/204262	5 235

RS-485	Yes	NI 9871 http://sine.ni.com/nips/cds/view/p/lang/sv/nid/204263	6 280
USB			

4.4.1 Conclusion

As the table above describes, there are certainly a market for WireFlow to expand into, if they decide to start developing these new types of modules. Although many of the leading networking standards are available for CompactRIO, standards as CC-Link, one of the largest open networking standard, and Sercos III are still not available for CompactRIO users.

4.5 Expanding the LabVIEW SDK

During the development of the SDK for LabVIEW we made some decisions in order to limit the time spent on the development process. Meaning that for other modules than the CANopen from the ABCC series the SDK would not work properly. To fix this and to give WireFlow a good estimation of how much there is left to do before the SDK is fully developed I decided to pinpoint the needed expansions.

4.5.1 Object support

The ABCC is, as explained before, built based on object orientation.

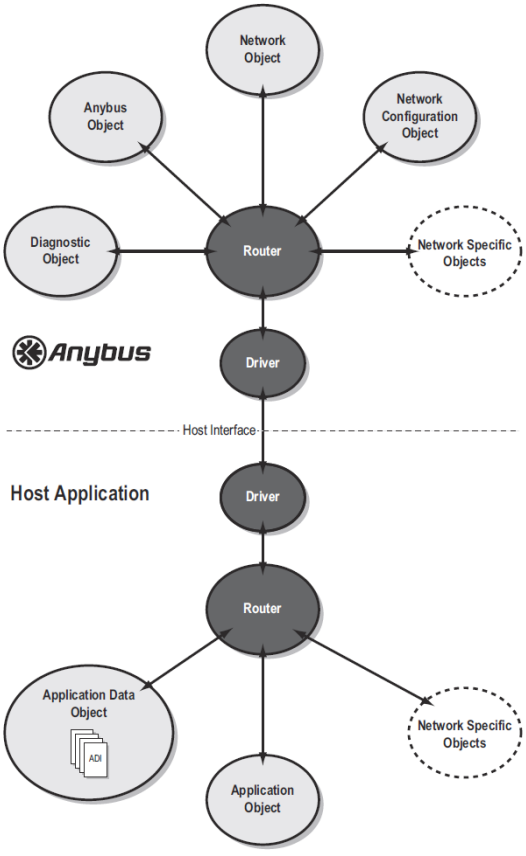


Figure 33 – ABCC object orientation [15, p. 7]

Every object in the Anybus "side" of the picture is implemented in the ABCC module and is, if supported by the host application, usable by the same. The "router" part is as explained in the software guide [15, p. 7]: *"The 'router'-part in the figure above symbolizes a software component which interprets the header information in an object message and routes its data to the appropriate software function."*
The router-part has been embedded into the driver and it is based on the same functionality as described above.

During this thesis the following implementation has been made:

- Basic driver with functionality for supporting the ABCC CANopen module
- Support for five Application Data Objects (ADI's).
- Support for CANopen network specific object.

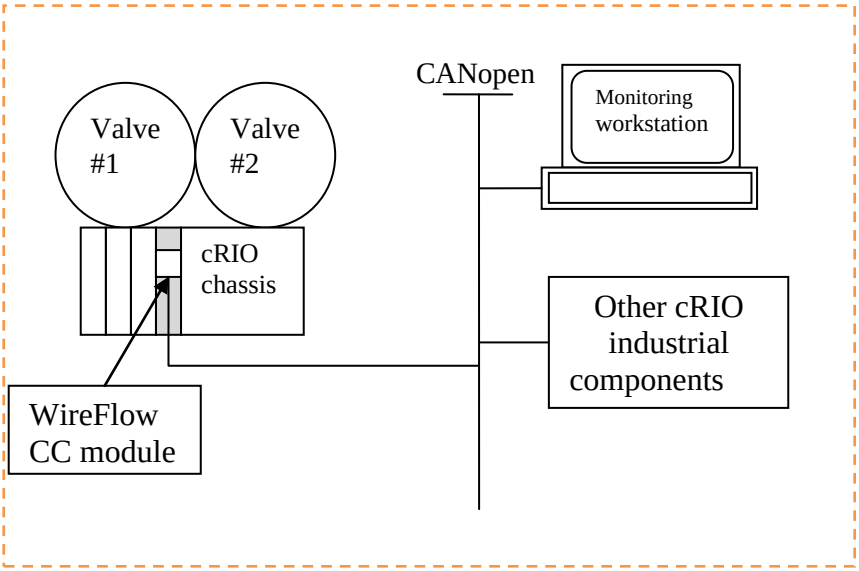
During the developing process the ABCC software design guide has been used as a basis. The developing only extends to the objects that are marked "mandatory" in the guide.

4.5.2 Still to be implemented:

- Make SDK independent regarding what ***data format*** that each module has, LSB or MSB.
- Take necessary action towards ***PDS*** (parameter data support) in case a module doesn't have it. If that is the case, what should happen?
- Add support for ***passive modules*** not supported at all in today's SDK.
- All ***network specific objects*** for the supported standards available in the ABCC series.
- Expanding the support for the ***Application Data Objects***, making it fully automatic.
- Implementing the ***Application Object***.
- Implementing support for the ***Diagnostics Object***.

4.5.3 Real-life scenario

In this part we will try to give the reader an understanding on how this future product can be used in a real-life scenario.
An industry develops some sort of electronics. In their industry they are about to invest in two valves for controlling the flow of some kind of liquid, these valves have a cRIO chassis integrated in them with IO's for controlling the valves.
The main network standard in the industry is CANopen, to which several machines and resources are connected to today, and now they want to be able to observe and control the valves using the same network.
By using a cRIO CANopen module from WireFlow the company can extend their current network and be able to control or observe the new valves all from one monitoring station.



This is an easy way of explaining how this new product can be implemented in expanding industries. Making sure that new equipment stays fully operational in the current networking standard instead of having to expand the network with another standard just to make sure an investment can be fully operational. It will be a lot easier to implement and also help the company save money.

References

[1] Bluetooth (2013). *Fast Facts.* http://www.bluetooth.com/Pages/Fast-Facts.aspx [2014-02-12].

[2] CAN in Automation (CiA) (2006). *CANopen.* http://www.can-cia.org/index.php?id=canopen [2014-02-12].

[3] CAN in Automation (CiA) (2006). *Controller Area Network (CAN).* http://www.can-cia.org/index.php?id=systemdesign-can [2014-02-12].

[4] CC-Link Partner Association (CLPA) (2013). *Welcome.* http://www.clpa-europe.com/ [2014-02-12].

[5] EtherCAT Technology Group (2012). *EtherCAT.* http://www.ethercat.org/en/ethercat.html [2014-02-12].

[6] IEEEXplore (IEEE). *An introduction to CANopen.* http://ieeexplore.ieee.org.lt.ltag.bibl.liu.se/xpl/articleDetails.jsp?tp=&arnumber=796056&queryText%3Dcanopen [2014-03-07].

[7] IXXAT (2008). *CANopen Basics – Introduction.* http://www.canopensolutions.com/english/about_canopen/about_canopen.shtml [2014-01-02]

[8] IXXAT (2008). *CANopen Basics – Device Configuration.* http://www.canopensolutions.com/english/about_canopen/device_configuration_canopen.shtml [2014-02-12].

[9] IXXAT (2008). *CANopen Basics – Process Data Exchange.* http://www.canopensolutions.com/english/about_canopen/pdo.shtml [2014-02-12].

[10] HMS (2011). *Anybus-CompactCom Mounting Kit Appendix*. Provided by HMS on CD.

[11] HMS [no date]. *Anybus-CompactCom Starter Kit Installation Guide.* Provided by HMS on CD.

[12] HMS (2012), CANopen module. [Picture], http://anybus.com/upload/334-Default-ABCC%20COP-2%20lma.jpg [2013-12-12].

[13] HMS (2012). *Introducing Anybus CompactCom – 30 series.* http://anybus.com/products/abcc30.shtml [2013-12-01].

[14] HMS (2013). *Anybus CompactCom Hardware Design Guide.* http://anybus.com/upload/Anybus-CompactCom-6318-Anybus_CompactCom_Hardware_design_guide.pdf [2014-01-02].

[15] HMS (2012). *Anybus CompactCom Software Design Guide*. http://anybus.com/upload/Anybus-CompactCom-9379-Anybus_CompactCom_Software_Design_Guide.pdf [2014-01-02].

[16] HMS (2010). *Anybus-CompactCom Standard Driver*. http://anybus.com/upload/Anybus-CompactCom-3403-Anybus-CompactCom-3295-Anybus%20CompactCom%20Driver%20Package%202.11.zip [2014-02-12].

[17] National Instruments (NI). 4 slot cRIO chassis NI 9075 [Picture]. http://sine.ni.com/images/products/us/02221111_m.jpg [2014-01-02].

[18] National Instruments (NI) (2013). *CompactRIO Third-Party Products.* http://www.ni.com/white-paper/2726/en/ [2014-01-02].

[19] National Instruments (NI). cRIO module. [Picture]. http://sine.ni.com/images/products/us/041119_crio9954_l.jpg [2014-01-02].

[20] National Instruments (NI) (2011). *NI cRIO-9951 CompactRIO Module Development Kit User Manual, Hardware User Manual*. Provided by WireFlow AB.

[21] National Instruments (NI). NI PXI 1031 Chassis. [Picture]. http://sine.ni.com/images/products/us/01051217_m.jpg [2012-12-12].

[22] National Instruments (NI) (2012). *What is LabVIEW?* http://www.ni.com/labview/ [2013-12-01].

[23] National Instruments (NI) (2010). *What is NI CompactRIO?.* http://www.ni.com/compactrio/whatis/ [2014-01-02].

[24] Open DeviceNet Vendors Association (ODVA) (2014). *ControlNet.* http://odva.org/default.aspx?tabid=244 [2014-02-04].

[25] Open DeviceNet Vendors Association (ODVA) (2014). *DeviceNet Technology Overview.* http://www.odva.org/Home/ODVATECHNOLOGIES/DeviceNet/DeviceNetTechnologyOverview.aspx [2014-02-12].

[26] Open DeviceNet Vendors Association (ODVA) (2006). *The Common Industrial Protocol (CIP) and the family of CIP Networks.* http://www.odva.org/Portals/0/Library/Publications_Numbered/PUB00123R0_Common%20Industrial_Protocol_and_Family_of_CIP_Netw.pdf [2014-02-12].

[27] PROFIBUS & PROFINET International (PI) (2012). *Profibus.* http://www.profibus.com/technology/profibus/overview/ [2014-02-12].

[28] Step Automation & Test (StepAT) [no date]. *Ethernet POWERLINK on CompactRIO.* http://www.stepat.com/index.php/content/download/3260/29835/version/1/file/Ethernet+POWERLINK+on+CompactRIO.pdf [2014-01-02].

[29] Universal Serial Bus (USB) (2014). http://www.usb.org [2014-02-12].

[30] WireFlow AB (WireFlow). *About.* http://www.wireflow.se/about-1 [2013-12-12].